Predicting the disease phenotypes of *de novo* ATP1A3 variants

Biren Dave
May 7 2018

1 Generating PROVEAN and SIFT scores for variants

The following code cell reads the spreadsheet containing the ATP1A3 mutations and generates an output file compatible with PROVEAN. `file_reader` reads an input `delim`-delimited file and generated an ordered dictionary.

PROVEAN requires input data to be space-delimited and formatted like this: `<Protein_ID> <position> <reference_residue(s)> <variant_residue(s)> <comment (optional)>`. Multiple variants can be analyzed at once if they are line-delimited. To achieve appropriate formatting, the `provean_input_formatter` function has been implemented. Finally, an output `.txt` file is generated by the function `output_writer`. This file can be directly uploaded to the online PROVEAN server.

```
In [1]: import csv

        NCBI_ref_id = "NP_689509.1"

        def file_reader(infile, delim):
            data = []
            with open(infile, "r") as fh:
                for row in csv.DictReader(fh, delimiter = delim):
                    data.append(row)
            return data

        def provean_input_formatter(l):
            provean_input = []
            for variant in l:
                temp = []
                if variant["type"] == "Missense_Mutation":
                    ref_aa = variant["substitution"][0]
                    alt_aa = variant["substitution"][-1]
                    pos = variant["substitution"][1:-1]
                    disease = variant["disease"]
                    temp = [NCBI_ref_id, pos, ref_aa, alt_aa, disease]
                    provean_input.append(temp)
                else:
                    pass
            return provean_input

        def output_writer(l, outfile, delim):
            with open(outfile, "w") as fh:
```

```
        writer = csv.writer(fh, delimiter = delim)
        writer.writerows(l)

variants = file_reader("mutations.csv", ",")
provean_input = provean_input_formatter(variants)
output_writer(provean_input, "provean_input.txt", " ")
```

2 Making a feature matrix

The tabulated PROVEAN output is then related back to the originial mutation data, so that disease outcome and protein domain data can be added. The dictonary `domains` maps certain amino acid positions to NCBI-annotated protein regions, `properties` maps single letter amino acid codes to a set of chemical properties, and `diseases` maps amino acid changes (in the format A[pos]B, where A is the reference residue and B is the altered residue) to disease phenotypes according to published data.

```
In [2]: '''
        NCBI_domains = {37: "phosphorylation site",
                   56: "phosphorylation site",
                   range(72,75): "interaction with phosphoinositide-3 kinase",
                   range(78,99): "transmembrane region",
                   range(122,143): "transmembrane region",
                   218: "phosphorylation site",
                   265: "phosphorylation site",
                   range(279,299): "transmembrane region",
                   range(311,329): "transmembrane region",
                   442: "phosphorylation site",
                   548: "phosphorylation site",
                   range(763,783): "transmembrane region",
                   range(834,857): "transmembrane region",
                   range(909,929): "transmembrane region",
                   933: "phosphorylation site by PKA",
                   range(942,961): "transmembrane region",
                   range(976,997): "transmembrane region"}
        '''

        pfam_domains = {range(33,102): "cation_ATPase_N",
                        range(153,345): "E1-E2_ATPase",
                        range(416,512): "cation_ATPase",
                        range(789,999): "cation_ATPase_C"}

        properties = {"R": "charged",
                      "K": "charged",
                      "D": "charged",
                      "E": "charged",
                      "Q": "polar",
                      "N": "polar",
                      "H": "polar",
                      "S": "polar",
                      "T": "polar",
                      "Y": "polar",
                      "C": "polar",
                      "W": "polar",
                      "A": "hydrophobic",
                      "I": "hydrophobic",
                      "L": "hydrophobic",
                      "M": "hydrophobic",
                      "F": "hydrophobic",
                      "V": "hydrophobic",
                      "P": "hydrophobic",
                      "G": "hydrophobic"}

        def get_domain(position):
            for key in pfam_domains.keys():
                if position in key:
                    return pfam_domains[key]
            return "unannotated domain"

        def get_chemical_properties(amino_acid):
            return properties[amino_acid]

        def prop_features(prop):
            if prop == "charged":
                return [1, 0, 0]
            elif prop == "polar":
                return [0, 1, 0]
            elif prop == "hydrophobic":
                return [0, 0, 1]

        def ncbi_domain_features(domain):
            if domain == "phosphorylation site":
                return [1, 0, 0, 0]
            elif domain == "interaction with phosphoinositide-3 kinase":
                return [0, 1, 0, 0]
            elif domain == "transmembrane region":
                return [0, 0, 1, 0]
            elif domain == "phosphorylation site by PKA":
                return [0, 0, 0, 1]

        def pfam_domain_features(domain):
            if domain == "cation_ATPase_N":
                return [1, 0, 0, 0, 0]
            elif domain == "E1-E2_ATPase":
                return [0, 1, 0, 0, 0]
            elif domain == "cation_ATPase":
                return [0, 0, 1, 0, 0]
            elif domain == "cation_ATPase_C":
                return [0, 0, 0, 1, 0]
            elif domain == "unannotated domain":
                return [0, 0, 0, 0, 1]


        def provean_sift_features(provean, sift):
            if provean < -2.5:
                p_pred = 1
            else:
                p_pred = 0
            if sift > 0.05:
                s_pred = 1
            else:
                s_pred = 0
            return [provean, p_pred, sift, s_pred]

        diseases = {}

        for variant in variants:
            diseases[variant["substitution"].strip()] = variant["disease"]

        provean_output = file_reader("provean_output.tsv", "\t")
```

Here, I define the custom object `Variant`, which contains a method `get_feature_vector` which constructs a 16-dimensional feature vector for that variant. These feature vectors are stored in the list `feature_vectors`, and their corresponding disease state is stored in the list `disease_list`.

```
In [3]: class Variant():
            def __init__(self, ref_aa, pos, alt_aa, provean, sift):
                self.ref_aa = ref_aa
                self.alt_aa = alt_aa
                self.pos = pos
                self.domain = get_domain(pos)
                #self.domain_features = pfam_domain_features(self.domain)
                self.ref_aa_prop = get_chemical_properties(ref_aa)
                self.alt_aa_prop = get_chemical_properties(alt_aa)
                self.provean = provean
                self.sift = sift
                try:
                    self.disease = diseases[ref_aa + str(pos) + alt_aa]
                except KeyError:
                    self.disase = "null"
            def get_feature_vector(self):
                aa_prop_vector = prop_features(self.ref_aa_prop) + prop_features(self.alt_aa_pro
                domain_vector = pfam_domain_features(self.domain)
                provean_sift_vector = provean_sift_features(self.provean, self.sift)
                return [self.pos] + aa_prop_vector + domain_vector + provean_sift_vector
            def get_disease(self):
                return self.disease
            def __str__(self):
                return "ref_aa: {}, alt_aa: {}, pos: {}, ref_aa_prop: {}, alt_aa_prop: {}, PROVE
                        self.ref_aa, self.alt_aa, self.pos, self.ref_aa_prop, self.alt_aa_prop,

        feature_vectors = []
        disease_list = []

        for var in provean_output:
            if var["PROTEIN_ID"] == NCBI_ref_id:
                v = Variant(var["RESIDUE_REF"], int(var["POSITION"]), var["RESIDUE_ALT"], float(
                feature_vectors.append(v.get_feature_vector())
                disease_list.append(v.get_disease())
```

3 Training the Support Vector Classifier (SVC)

```
In [4]: from sklearn import svm

        classifier = svm.SVC()
        classifier.fit(feature_vectors, disease_list)

Out[4]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
```

4 Making predictions

```
In [5]: pos = input("enter position of the amino acid variant: ")
        ref_aa = input("enter the single letter code of the reference amino acid: ")
        alt_aa = input("enter the single letter code of the altered amino acid: ")
        provean = input("enter the PROVEAN score of the variant: ")
        sift = input("enter the SIFT score of the variant: ")

In [6]: v = Variant(ref_aa, int(pos), alt_aa, float(provean), float(sift))

        print("predicted disease phenotype: " + classifier.predict([v.get_feature_vector()])[0])

predicted disease phenotype: AHC
```